Experimental Evaluation of Jitter Buffer Algorithms on Voice over IP Networks

J.P.Ouedraogo, L.Sun and I.H.Mkwawa

Signal Processing and Multimedia Communications,
University of Plymouth, Plymouth, United Kingdom
e-mail: L.Sun@plymouth.ac.uk

Abstract

In Voice over IP (VoIP) applications, speech quality is mainly affected by network impairments such as delay, jitter and packet loss. Jitter buffer algorithms are used to lessen the impact of jitter by delaying the packet playout time. The aim of the paper is to evaluate and enhance the performance of different jitter buffer algorithms over a VoIP testbed based on Gstreamer. Preliminary results show that a combination of an algorithm that reacts more quickly to network variations and an algorithm that reduces the playout delay achieves the best trade-off between playout delay, buffer discarding rate and perceived listening only speech quality.

Keywords

Voice over IP; Jitter buffer algorithm; Gstreamer; Listening only speech quality

1 Introduction

In Voice over Internet Protocol (VoIP), one of the main challenges for applications is to smoothly deliver voice samples despite network impairments such as delay, packet loss and jitter. To deal with those situations, a jitter buffer can be used to achieve a better overall speech quality for the user. Hence, each arriving packet is held in a buffer, and its real playout time is delayed to allow late packets to reach the receiver on time (i.e. before the delayed scheduled time).

Two major approaches can be noticed, fixed jitter buffer which does not adapt through time and adaptive jitter buffer which takes network variations into account. The latter can be divided further into buffer that adjusts at the beginning of talkspurts (Ramjee *et al.* 1994) or during talkspurt (Liang *et al.* 2001). This paper focuses on adaptive jitter buffer, as more algorithms and computations are involved.

In terms of jitter buffer, a trade-off must be found between the delay added by the buffer and the loss due to packets late arrival.

Several papers have been published in relation with the efficiency of Ramjee *et al.* (1994) algorithm, such as Moon *et al.* (1998) and Roccetti *et al.* (2001) but none of them investigates the impact of its algorithm on Gstreamer.

Hence, the aim of this paper is firstly to evaluate the jitter buffer of Gstreamer, and the impact of the fourth algorithm presented in Ramjee *et al.* (1994), and secondly to enhance the efficiency of the latter in terms of playout delay, buffer discarding rate and perceived listening only speech quality.

The remainder of the paper is structured as follows. Section 2 presents the research method and experimental design. Section 3 describes the algorithms used. Section 4 explains the findings and analyses. Section 5 concludes the paper.

2 Research method and experimental design

The jitter buffer algorithms assessment is performed under a range of jitter values between 10ms and 60ms (as suggested by Calyam and Lee, 2005). Nevertheless, the paper only present the significant results obtained at 60ms.

Figure 1 shows the test bed set up and the following explains the main components used in this study. An audio file (ITU-T English female voice sample, 2009) is uploaded onto a VLC server (PC 1) so it can be retrieved on demand. NetEm, a Linux command line network emulator tool, is used to control the jitter under a normal (Gaussian) distribution. It has been configured to operate on incoming traffic at the receiver end, just after the network device. Finally, Gstreamer (including "Gstreamer Bad Plugins" that contains the jitter buffer algorithm), a streaming media application framework (PC 2), retrieves the audio file from the VLC server, using RTSP protocol. Gstreamer has been chosen because it is an open source product, and it is used in UCT IMS Client (Waiting *et al.* 2009), a VoIP client in development. Lame (2009) is used to encode an audio file in MP3, using variable bitrate. MP3 format is widely used as music on hold source files in Asterix IP-PBX (2009).

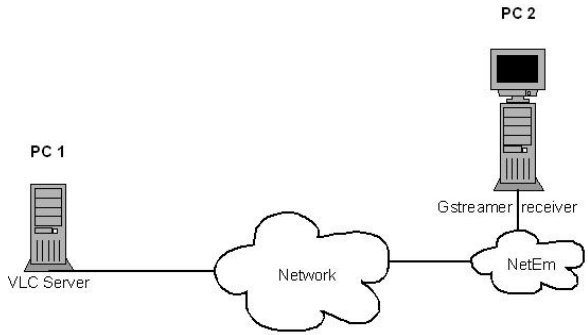

Figure 1: Test bed diagram

The variable bitrate (VBR) algorithm analyses each segment of the file, and reduces its bitrate (thus its size) according to its complexity. Hence, a silence segment will occupied less space than a talkspurt. As a result, algorithms that adapts according to talkspurt and silence can base their calculations on the size of the packet they process.

The length of the ITU-T sample has firstly been elongated from originally 8 to 16 seconds by duplicating the original sample to have enough data to analyse. The audio file was originally in WAV format. As it was not possible with VLC to retrieve the WAV file on demand (only live streaming), the file has been encoded in MP3.

The file retrieved with Gstreamer is stored in MP3, thus it is further encoded in WAV to allow the comparison with the original sample. The re-encoding lessens the audio quality, but the difference is very small: a loss of 2.47% of the quality has been noticed (by comparing two files PESQ score under no network impairment). Thus, this loss of quality is considered as negligible in the scope of this study.

The ITU-T standard P.862 (2001) Perceptual Evaluation of Speech Quality (PESQ) will determine the quality of the degraded audio file. PESQ is an automated tool for the assessment of speech quality, as a human being would perceive it. It is a "full reference" algorithm: it uses a reference test signal, and compares it against a degraded signal. The result is mapped on a scale of 1 to 4.5.

According to Opticom (2008), "the delay between the reference and the test signal is no longer constant". PESQ has then be developed to get over this issue, as algorithms that control the jitter buffer are aimed to change the playout time, i.e. the delay between the original file and the test file. This task is achieved by a "time alignment algorithm" (Opticom, 2008).

A PESQ C language code version defined in ITU-T recommendation P.862 (2001) will be used in this study.

3 Jitter Buffer algorithms

3.1 Implemented algorithms

The paper focuses on adaptive jitter buffer, as fixed cannot adapt to network variations. The notations used to describe the algorithms are explained in Figure 2. For packet i, t_i and p_i are respectively the send time and the playout time, d_i is the mean estimated network delay, v_i is the estimated variation of network delay, and n_i is the actual network delay.

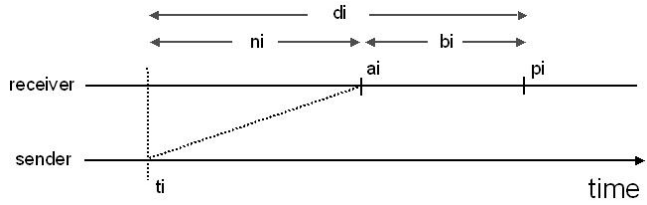

Figure 2 Timing associated with packet *i*

The implemented algorithms are adapted from Ramjee *et al.* (1994). The following one is called Algorithm 0 in this paper, as it is the strict implementation of Ramjee *et al.*'s algorithm. The playout time of the first packet in a talkspurt is computed as:

$$p_i = t_i + \hat{d}_i + \alpha * \hat{v}_i \quad (1)$$

where $\alpha = 4$. $\hat{d}_i$ and $\hat{v}_i$ (running estimates of the mean and variation of network delay, respectively d_i and v_i) are calculated as:

$$\hat{d}_i = \beta * \hat{d}_{i-1} + (1 - \beta) * n_i \text{ and } \hat{v}_i = \beta * \hat{v}_{i-1} + (1 - \beta) * |\hat{d}_i - n_i| \quad (2)$$

where $\beta = 0.875$.

For any further packet that belongs to the same talkspurt, the playout time is computed as:

$$p_j = p_i + t_j - t_i \quad (3)$$

To reduce the large average playout time noticed by Roccetti *et al.* (2001), six algorithms based on Algorithm 0 have been assessed. The computation of p_i for Algorithms 1 to 5 is presented in Table 1. For Algorithms 3 and 5, the weighting factors of $\hat{d}_i$ have been inverted to react more quickly to network fluctuations:

$$\hat{d}_i = (1 - \beta) * \hat{d}_{i-1} + \beta * n_i \quad (4)$$

	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5
p_i	$t_i + \frac{\hat{d}_i}{2} + 2\hat{v}_i$	$t_i + \frac{\hat{d}_i}{4} + \hat{v}_i$	$t_i + \hat{d}_i + \hat{v}_i$	$t_i + \frac{\hat{d}_i}{6} + \frac{\hat{v}_i}{2}$	$t_i + \frac{\hat{d}_i}{4} + \hat{v}_i$

Table 1 Computation of p_i for Algorithms 1 to 5

The logic behind Algorithms 1, 2 and 4 is to reduce the playout time and investigate to what extent it affects the perceived speech quality. Algorithm 3 investigates the influence of $\hat{d}_i$ when it is set to attach more importance to the latest network delay variation. Finally, Algorithm 5 is a combination of Algorithms 2 and 3, based on their performances (see Section 4).

3.2 Jitter buffer in Gstreamer

"Gstreamer Bad Plugins" (2009) includes a jitter buffer algorithm. It will be compared to six algorithms implemented in the same structure. Gstreamer uses two algorithms adapted from Fober *et al.* (2005). The first one, a parabolic weighting factor algorithm, is applied during the two first seconds (or the 512 first packets). The following one is known as "window low point averaging". The calculation is computed as:

$$New\ skew = \frac{MIN\left[\left(a_j - a_i\right), \left(t_j - t_i\right)\right] + 124 * previous\_skew}{125} \quad (5)$$

Where a_j and a_i are respectively the current packet arrival time and the first packet arrival time; t_j and t_i are respectively the current packet send time and the first packet send time (see Figure 2); MIN is a function that outputs the lowest number of its arguments.

This skew is added to the RTP timestamp, so the final playout time of packet *j* is:

$$p_j = a_i + \left(t_j - t_i\right) + skew \quad (6)$$

4 Jitter buffer performance analysis

As NetEm generates random numbers to emulate jitter, it produces a different trace each time it is used. The mean of observed parameters is computed for each set of traces (four repetitions for Algorithm 0, fifty repetitions for Algorithms 1 to 5, as proposed by Blatnik *et al.* 2006) and presented in Figure 3 and Figure 4. Although Algorithm 0 could be assessed along with Algorithms 1 to 5, its playout time is too large (30% more than Gstreamer original algorithm) so it cannot be considered as a good trade-off between the delay added by the buffer and the loss due to packets late arrival.

Algorithms 1 and 2 PESQ scores are close (respectively 3.04 and 3.07 on average), even if their delay have been divided by respectively 2 and 4. However, it can be noticed that Algorithm 4 (which divided p_i arguments by 6) PESQ score drops significantly (2.73) in comparison with Algorithms 1 and 2. Hence, it could be concluded that Algorithm 4 computation is not optimum in terms of PESQ score. Algorithm 5, which is partly composed of Algorithm 2, has approximately the same PESQ score of the latter (2.99), instead of the worse Algorithm 3 PESQ score (which compose the second part of Algorithm 5).

In terms of buffer discarding rate, Algorithm 5 behaves the same way as Algorithms 2 and 3 (respectively 13.51%, 12.35% and 13.40%), and discards a similar number of packets. Algorithm 1 discards fewer packets (9.20%) but also has the highest average playout delay (far more than the other algorithms).

In addition, the relation between PESQ scores and buffer discarding rates can be analysed by contrasting the two graphics of Figure 3. Algorithms 1, 2 and 4 behaviours were expected: the greater the PESQ score, the lower the buffer discarding rate. Furthermore, Algorithms 2 and 5 have approximately the same PESQ scores and the same buffer discarding rates.

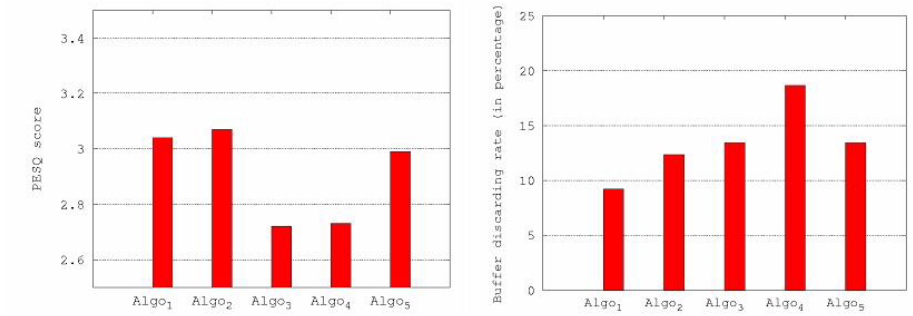

Figure 3 PESQ score and buffer discarding rate (in percentage) for Algorithms 1 to 5

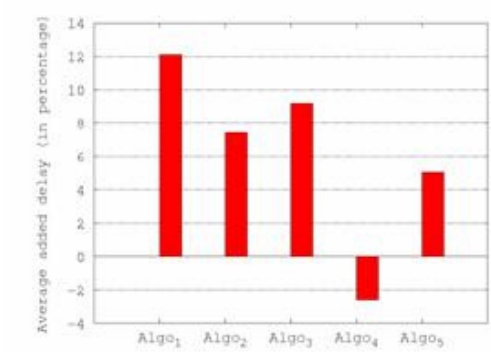

Figure 4 Average added delay (in percentage) for Algorithms 1 to 5

However, an unexpected behaviour can be noticed for Algorithm 3. Indeed, Algorithm 3 buffer discarding rate is almost the same as Algorithms 2 and 5 buffer discarding rate, but Algorithms 2 and 5 PESQ scores are better than Algorithm 3 PESQ score. Algorithms are based on packet sizes to detect talkspurt and silence periods. Hence, some beginnings of talkspurt or silence parts can be misunderstood by the algorithm, if for instance three contiguous packet sizes arrive with the following distribution: $\{x\ ;\ y\ ;\ x\}$ where $\{x\}$ is a packet size that represents a talkspurt and $\{y\}$ is a packet size that represents a silence. In this case, the algorithm will set the second $\{x\}$ as the beginning of a new talkspurt and may change the playout time, especially in the case of Algorithm 3, and the new one can overlap a previous packet that contains a talkspurt. Hence, this could be the explanation of the relation between the PESQ score and the buffer discarding rate noticed above for Algorithm 3. Moreover, the green rectangles in Figure 5 show a lack of several packets at the beginning of the second waveform comparing with the original sample, which confirms this assumption. This behaviour does not happen when a real talkspurt is detected, as highlighted by the red circles in Figure 5, where the new playout time set by the algorithm at the beginning of the talkspurt overlaps the previous silence packets.

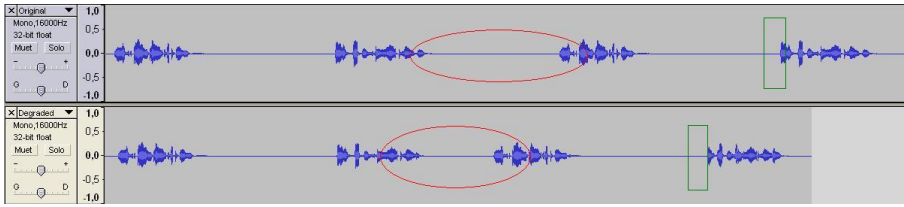

Figure 5 Variable delay (for Algorithm 3)

The original $\hat{d}_i$; d_i (in Algorithm 0) reacts slowly to network fluctuations, as the weight factors are 0.125 for the new n_i, and 0.875 for the previous $\hat{d}_i$. By inverting those weighting factors in Algorithm 3, the "memory" of the algorithm $\hat{d}_{i-1}$ is reduced for the advantage of the latest n_i calculation. Hence, $\hat{d}_i$ will vary more quickly. This behaviour is clearly noticeable in Figure 5, where the red circle on the first sample (the original one) corresponds to the original silence length, whereas the second red circle below corresponds to the degraded silence length.

Finally, Algorithm 5 is a combination of Algorithms 2 and 3. Algorithm 5 has overcome the issue addressed by Algorithm 3: the variable delay between two talkspurts is not as larger as Algorithm 3. Its playout delay is also reduced, compared with Algorithms 1 and 2. In terms of buffer discarding rate, although it drops more packets than Algorithm 1, this percentage can be compared to Algorithm 2 as they have the same p_i calculation. By applying the findings of Algorithm 3 to Algorithm 5, the latter reacts more quickly to network variations.

5 Conclusion and future work

This study pointed out the significance of the trade-off between jitter, buffer discarding rate and perceived quality speech. A test bed composed of two machines, a sender (VLC server) and a client (Gstreamer pipeline) has been set up to study the performance of several jitter buffer algorithms under jitter values from 10ms to 60ms.

It appears that Algorithm 5, which is a combination of an algorithm that reacts more quickly to network variations (Algorithm 3) and an algorithm that reduces the time at which the packet is played at the client side (Algorithm 2) generates the best overall results when it comes to find a good trade-off between jitter, buffer discarding rate and perceived quality speech.

Those findings can be carried out further, and it could be interesting to evaluate the performance of Gstreamer and the modified code in a "real" VoIP environment. Hence, the implementation of the modified Gstreamer code in a VoIP client that can handle a communication in both ways, such as UCT IMS Client (Waiting *et al.* 2009) could be a really attracting project.

6 References

Asterix Web Site (2009) “The Open Source PBX & Telephony paltform”, http://www.asterisk.org/, (Accessed 08 September 2009).

Blatnik, R., Kandus, G., Javornik, T. (2006) *Experimental test-bed for VoIP/VoWLAN voice quality measurements*, Proceedings of the 4th WSEAS International Conference on Electromagnetics, Wireless and Optical Communications, 2006, World Scientific and Engineering Academy and Society, pp5-10, ISSN:1790-5095.

Calyam, P., Lee, C.G. (2005) *Characterizing voice and video traffic behavior over the Internet*, International Symposium on Computer and Information Sciences (ISCIS), Proceedings published by Imperial College Press in a special edition of "Advances in Computer Science and Engineering".

Fober, D., Orlarey, Y. and Letz, S. (2005) *Real Time Clock Skew Estimation over Network Delays*, Laboratoire de recherche en informatique musicale, Grame, France.

GStreamer Web Site (2009), “Bad Plugins 0.10 Plugins Reference Manual”, http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-plugins/html/gst-plugins-bad-plugins-gstrtpjitterbuffer.html, (Accessed 17 February 2009).

ITU-T Web Site (2009), “Test Signals for Telecommunication Systems, Annex B speech files”, http://www.itu.int/net/itu-t/sigdb/genaudio/AudioForm-g.aspx?val=10000501, (Accessed 15 January 2009).

ITU-T Web Site (2001), “ITU-T Recommendation P.862”, http://www.itu.int/rec/T-REC-P.862/, (Accessed 15 January 2009).

Lame documentation Web Site (2009) “Guide to command line options”, http://lame.cvs.sourceforge.net/*checkout*/lame/lame/USAGE, (Accessed 05 May 2009).

Liang, Y., Färber, N. and Girod, B. (2001) *Adaptive playout scheduling using time-scale modification in packet voice communications*, Proceedings of the Acoustics, Speech and Signal Processing, Volume 03, 2001, IEEE Computer Society, pp1445-1448, ISBN: 0-7803-7041-4.

Moon, S.B., Kurose, J. and Towsley, D. (1998) *Packet audio playout delay adjustment: performance bounds and algorithms*, Multimedia Systems, Volume6, Issue 1, Springer-Verlag New York, Inc, pp17-28, ISSN:0942-4962.

Opticom Web Site (2008), “Voice quality testing: PESQ”, http://www.opticom.de/technology/pesq.html, (Accessed 12 March 2009).

Ramjee, R., Kurose, J., Towsley, D. and Schulzrinne, H. (1994) “Adaptive playout mechanism for packetized audio applications in wide-area networks”, *Proceedings of IEEE INFOCOM*, Volume 2, 1994, pp680–688, ISBN: 0-8186-5570-4.

Roccetti, M., Ghini, V., Pau, G., Salomoni, P. and Bonfigli, M.E. (2001) *“Design and Experimental Evaluation of an Adaptive Playout Delay Control Mechanism for Packetized Audio for Use over the Internet”*, *Multimedia Tools and Applications*, Volume 14, Issue 1, 2001, Kluwer Academic Publishers, pp23-53, ISSN:1380-7501.

Waiting, D., Good, R., Spiers, R., Ventura, N. (2009) *The UCT IMS Client*, 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops, TridentCom 2009, pp1-6, ISBN: 978-1-4244-2846-5.